Dokumentation Gruppe 4

**Thema: üK 223**

**Dokumentinformationen**

Dateiname: Dokumentation Gruppe 4.docx

Speicherdatum: 05.11.2021

**Autoreninformationen**

Autor: Anojan Pirabakaran | Enes Spahiu

E-Mail: anojan.pirabakaran@noseryoung.ch | enes.spahiu@noseryoung.ch

**Inhaltsverzeichnis**

[1 Einleitung 4](#_Toc88032395)

[1.1 Sinn und Zweck 4](#_Toc88032396)

[2 Aufgabenstellung 5](#_Toc88032397)

[2.1 Funktionale Anforderungen 5](#_Toc88032398)

[2.2 Nicht funktionale Anforderungen 5](#_Toc88032399)

[2.2.1 Implementation 5](#_Toc88032400)

[2.2.2 Testing 5](#_Toc88032401)

[2.2.3 Multiuserfähigkeit 5](#_Toc88032402)

[2.2.4 Dokumentation 6](#_Toc88032403)

[3.1 Github Repository 7](#_Toc88032404)

[3.2 Berechtigungsmatrixen 7](#_Toc88032405)

[3.2.1 Home-Seite 7](#_Toc88032406)

[3.2.2 Gruppen 7](#_Toc88032407)

[4 UML 8](#_Toc88032408)

[4.1 Domänenmodell 8](#_Toc88032409)

[4.2 Klassen-Diagramm 9](#_Toc88032410)

[4.3 Use-Cases 10](#_Toc88032411)

[4.4 Sequenz-Diagramm 12](#_Toc88032412)

[5 Endpoints 13](#_Toc88032413)

[5.1 findAll 13](#_Toc88032414)

[5.2 findById 13](#_Toc88032415)

[5.3 getAllUsersOfGroup 14](#_Toc88032416)

[5.4 delete 14](#_Toc88032417)

[5.5 postMethod 15](#_Toc88032418)

[5.6 updateGroup 15](#_Toc88032419)

Änderungsgeschichte

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Datum** | **Autor** | **Details** |
| 1.0 | 05.11.2021 | Anojan | Dokument erstellt |
| 1.1 | 10.11.2021 | Anojan | Grundlagen der Doku aufgestellt |
| 1.2 | 12.11.2021 | Enes | Diagramme hinzugefügt |

# Einleitung

## Sinn und Zweck

Das vorliegende Dokument beschreibt unser Abschlussprojekt im üK Modul 223. In diesem Modul geht es, um Multi-User-Applikationen. Wir werden unseren Schwerpunkt auf Spring Boot in Verbindung mit einer Datenbank setzen.

# Aufgabenstellung

Im Folgenden wird die Aufgabenstellung erklärt.

## 2.1 Funktionale Anforderungen

Unsere Aufgabe als Team 4 ist es ein Gruppen-System zu implementieren, bei dem man User in eine Gruppe einteilen kann und die User zusammen mit den Admins haben dann auch als einzige Zugriff auf die Daten der Gruppe.

Eine Bedingung ist, dass ein User nur in maximal einer Gruppe sein kann, ist er also schon in einer Gruppe darf er nicht auch noch in einer anderen Gruppe sein. Die andere Bedingung besagt, dass eine Gruppe nur von einem Admin erstellt und verwaltet werden kann, also jemandem mit Admin rechten.

Nur User mit Adminrechten oder User, die in der entsprechenden Gruppe sind, können Informationen der Gruppen anschauen, verwalten und löschen.

Mithilfe von Pagination, sollten wir Endpoints in unserem Programm implementieren, um alle Mitglieder einer Gruppe anzuzeigen.

Die einzelnen Gruppen brauchen Informationen, wie Mitglieder, Gruppenname, Motto etc.

## 2.2 Nicht funktionale Anforderungen

## 2.2.1 Implementation

Die einzelnen Daten müssen in einem PostgreSQL Datenbank speichert und mit OR-Mapping und JPA realisiert werden.

## 2.2.2 Testing

Ein Backend-Endpoint muss definitiv getestet werden. Funktionalitäten, die wir implementiert haben, müssen mit Component-Tests in Post getestet werden. Die Endpoints werden mit mehreren Usern und Rollen getestet und mindestens ein Erfolgsfall und ein Error Fall muss getestet werden.

## 2.2.3 Multiuserfähigkeit

Die ACID-Prinzipien werden eingehalten.

## 2.2.4 Dokumentation

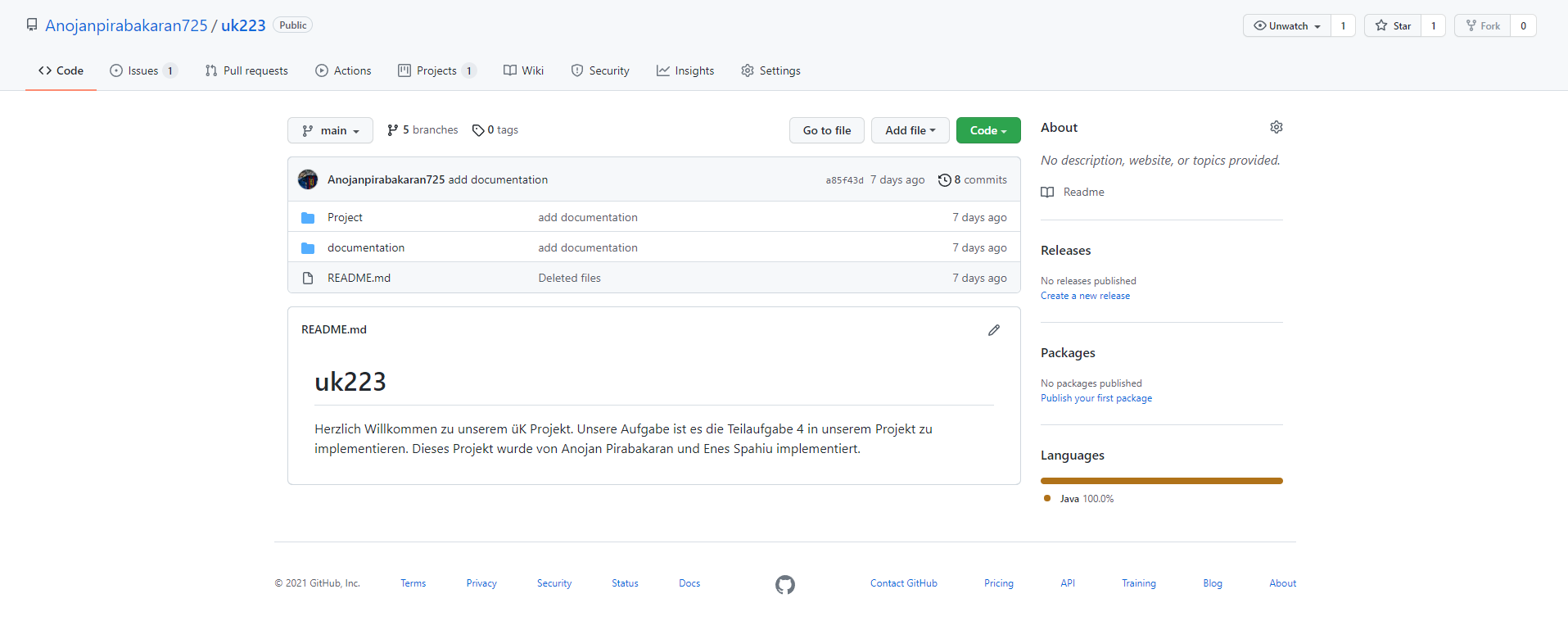
Die Funktionen werden genau und verständlich dokumentiert. Die Dokumentation erhält:

* ein Readme File im Repo
* ein Domänenmodell
* ein Klassen-Diagramm
* ein Sequenz-Modell
* Use Case

1. Administratives

Im folgenden Abschnitt wird gezeigt was unsere Administrativen Schritte waren, bevor wir an das wirkliche Implementieren gegangen sind.

## 3.1 Github Repository



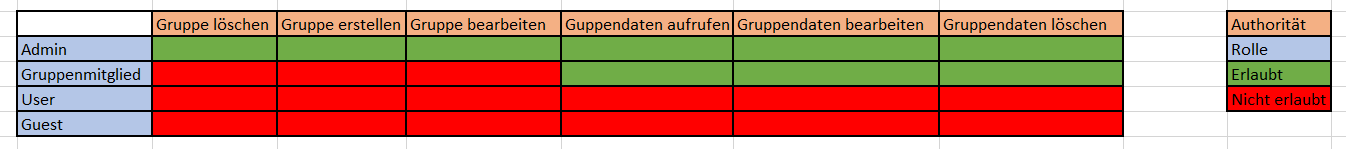
## 3.2 Berechtigungsmatrixen

### 3.2.1 Home-Seite

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung

### 3.2.2 Gruppen



# UML

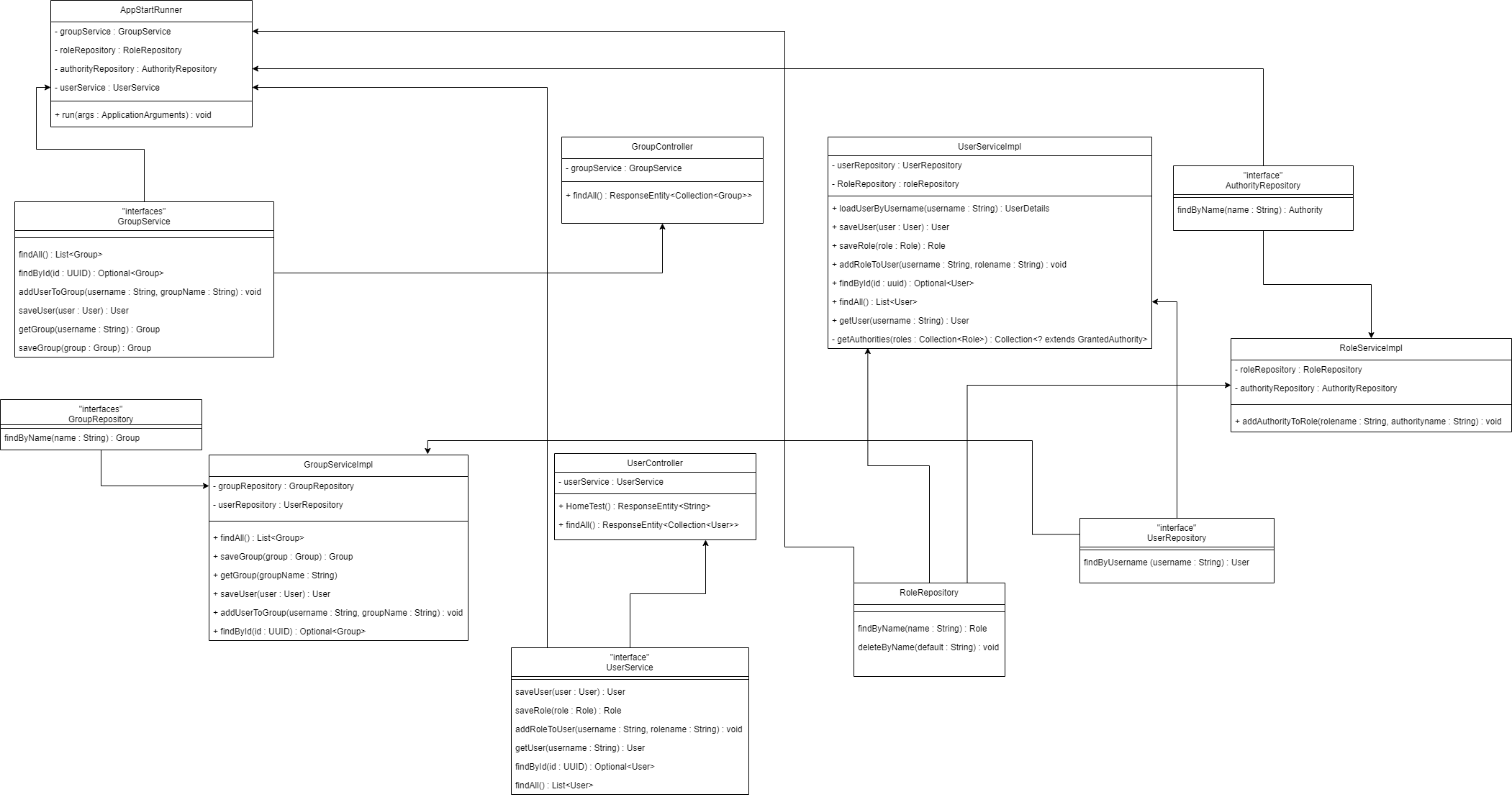
Im Folgenden werden unsere UMLs gezeigt.

## 4.1 Domänenmodell

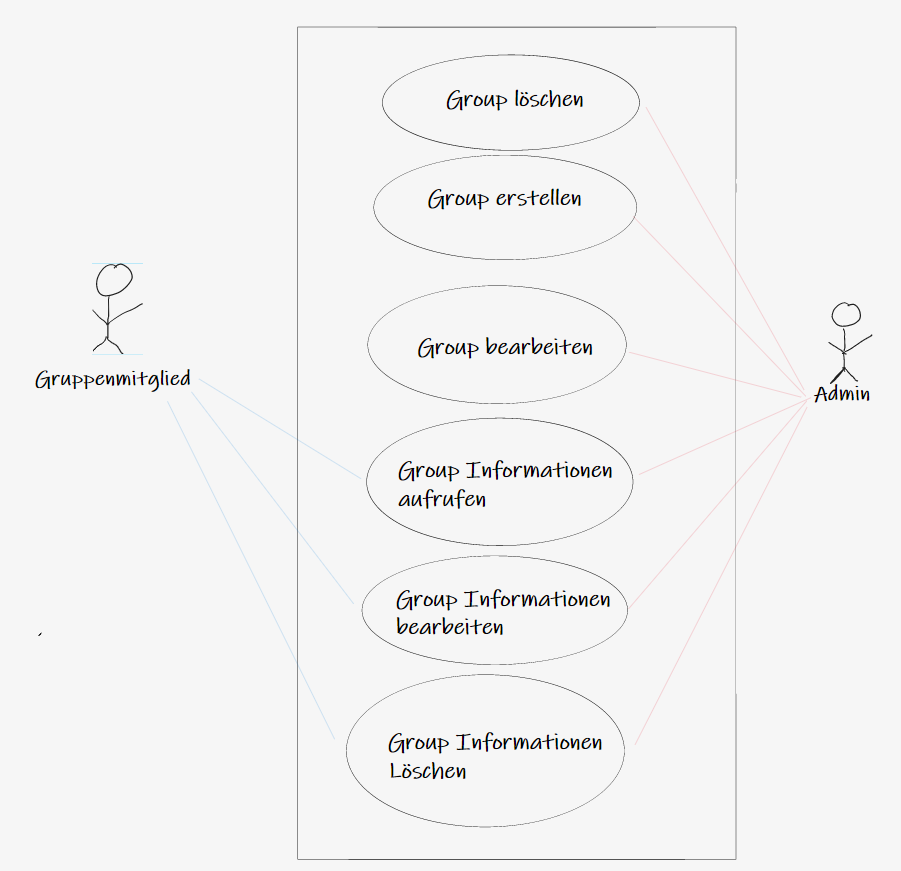
Ein Bild, das Text, drinnen, Whiteboard enthält.

Automatisch generierte Beschreibung

## 4.2 Klassen-Diagramm



## 4.3 Use-Cases



|  |  |
| --- | --- |
| **Use Case:** | Gruppe hinzufügen (Post-Endpoint) |
| **Use Case ID:** | 1 |
| **Kurzbeschreibung:** | Kunde erstellt eine neue Gruppe |
| **Vorbedingung:** | Kunde sollte gemäss dem JSON-File die Daten eingeben |
| **Akteur (Primary):** | User |
| **Akteur (Secondary):** | Keiner |
| **Hauptablauf:**   1. Der Use Case startet, wenn der User einen Post-Request auf den Endpoint «postMethod» schickt, um eine neue Gruppe zu erstellen 2. WHILE (SOLANGE) die Angaben nicht korrekt sind 3. Wird das System eine «500 Internal Server Error» ausgeben 4. Man gibt noch mal all mit den korrekten Angaben eingeben 5. Das System überprüft diese Eingaben 6. Die Gruppe wird in der Datenbank erstellt. | |
| **Nachbedingung:** | Eine neue Gruppe wird erstellt |
| **Alternative Flows:**  Doppelter Gruppenname  Ungültiger UUID | |

|  |  |
| --- | --- |
| **Use Case:** | Gruppe bearbeiten (Put-Endpoint) |
| **Use Case ID:** | 2 |
| **Kurzbeschreibung:** | Kunde ändert Informationen der Gruppe |
| **Vorbedingung:** | Kunde sollte gemäss dem JSON-File die Daten eingeben |
| **Akteur (Primary):** | Kunde |
| **Akteur (Secondary):** | Keiner |
| **Hauptablauf:**   1. Der Use Case startet, wenn der User einen Put-Request auf «updateGroup» schickt, um die Informationen zu ändern 2. WHILE (SOLANGE) die Angaben nicht korrekt sind 3. Wird das System eine «500 Internal Server Error» ausgeben 4. Kunde muss die Angaben ändern 5. Das System überprüft diese Angaben 6. Die Daten wurden geändert | |
| **Nachbedingung:** | Die Daten der Gruppe wurden geändert |
| **Alternative Flows:**  Es gibt keine Gruppe mit der angegeben UUID  Gruppenname existiert schon | |

## Sequenz-Diagramm

# Endpoints

Im Folgenden werden unsere Endpoints beschrieben, also was sie machen und welchen Fällen sie eingesetzt werden.

## 5.1 findAll

Findet alle Gruppen, die es gibt und gibt ihren Attributen aus. Im Postman sieht das so aus.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

## 5.2 findById

Findet die gesuchte Gruppe anhand ihrer ID und gibt ihren Attributen aus. Im Postman sieht das so aus.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

## 5.3 getAllUsersOfGroup

Findet die Gesuchte Gruppe anhand ihrer ID und gibt ihre Attributen sowie ihre User aus. Es wird Pagination verwendet, was man auch in der URL mitgeben muss. Im Postman sieht das so aus.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

## delete

Findet die Gesuchte Gruppe anhand ihrer ID und löscht sie dann von der Datenbank. Im Postman sieht das so aus.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Wenn man versucht eine ID einzugeben, die es nicht gibt, dann wirft das Programm eine InstanceNotFoundException

## postMethod

Man gibt per Postman den Body mit, die ID wird automatisch erstellt. Im Postman sieht das so aus.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

## updateGroup

Findet die gesuchte Gruppe per ID, dann kann man den Attributen der Gruppe verändern. Im Postman sieht das so aus.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung